Modelling with FTS: a Collection of Illustrative Examples

Classen, Andreas

Publication date:
2010

Document Version
Early version, also known as pre-print

PReCISE – FUNDP
University of Namur
Rue Grandgagnage, 21
B-5000 Namur
Belgium

TECHNICAL REPORT January 18, 2010

AUTHORS A. Classen
APPROVED BY P. Heymans
EMAILS {acs}@info.fundp.ac.be
STATUS Addendum to the paper *Model Checking Lots of Systems: Efficient Verification of Temporal Properties* appearing in the proceedings of *ICSE 2010 32nd International Conference on Software Engineering*, Cape Town, South Africa.
REFERENCE P-CS-TR SPLMC-00000001
PROJECT MoVES
FUNDING FNRS, the Walloon Region, Interuniversity Attraction Poles Programme of the Belgian State of Belgian Science Policy

Modelling with FTS: a Collection of Illustrative Examples

Modelling with FTS: A Collection of Illustrative Examples

Andreas Classen*
PReCISE Research Centre,
Faculty of Computer Science,
University of Namur
5000 Namur, Belgium
acs@info.fundp.ac.be

1 Introduction

FTS, featured transition systems, are a formalism designed to describe the combined behaviour of a whole system family [3]. FTS are transition systems [1, 2] (TS in short) in which transitions are labelled with *features* of a software product line [4] (in addition to being labelled with actions). This allows to model very detailed behavioural variations of the product line. In addition, features as treated as first-class abstractions, which allows both explicit variability management and separation of concerns, since a global view of the variability is available in a feature diagram (FD in short).

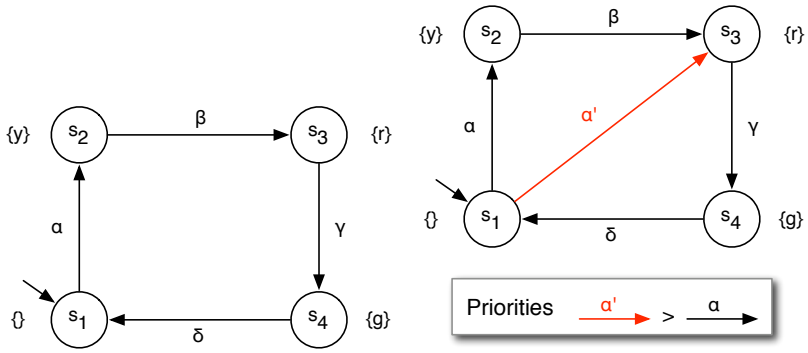
FTS come with a tool-supported model checking approach that allows to verify FTS against LTL properties.[1] The purpose of the approach is to verify all the products of a family at once and to pinpoint the products that violate properties. An empirical evaluation showed substantial gains over individual product verification [3].

We report here on a study of examples found in the literature that we did in order to evaluate our approach. The examples include the beverage vending machine from [5] in Section 3, the wiper system from [6] in Section 4, and the mine pump controller [7] in Section 5. We start with an introductory example, the red lights, in Section 2.

2 The red lights

Let us start with the red lights system, a classic among introductory examples in model checking. In its basic version, the red lights controller will switch between

*FNRS Research Fellow.
[1] Download at www.info.fundp.ac.be/~acs/fts



(a) The basic version without variability. (b) With one additional feature.

Figure 1: FTS of the red light controller.

yellow, red and green as shown by the TS in Figure 1(a). Initially the light is off, represented by an empty labelling of the initial state s_1; it then switches to yellow in s_2 (the y label), then to red and then to green. The action names here are not important and were thus named α through δ. So far, the controller can be modelled with a conventional TS.

There exists another version of the controller, which omits the yellow light and immediately shows red. In order to model the second version, one could easily draw a second TS without state s_2 and a transition from s_1 to s_3—and so forth for every version. In general, however, such an approach would not scale, since the number of different versions can become large. This is despite the fact that these versions generally only differ in small details.

In consequence, we proposed FTS [3], an extension of classical TSs, where transitions can be labelled with features[2] drawn from a variability model such as an FD [8]. To model the second red light variant in FTS, it is sufficient to add a transition from s_1 to s_3, to label it with a different feature, say SkipYellow, and to document the fact that there are two variants (one with SkipYellow and one without) in a variability model. The new transition α' also has to be of higher priority than α, otherwise, the variant with SkipYellow would still have the α transition.

Once an FTS of the system exists, it can be verified using the model checking algorithm proposed in [3]. We focus here on the modelling aspects and will thus not go into details about model checking. In short, it will check a temporal property for all systems represented by the FTS in one shot. In case the property is violated, the algorithm will pinpoint the products that violate it.

[2]Feature labels are represented by colouring in this report.

3 The vending machine [5]

The vending machine example originally appeared in [5] to illustrate the use of modal transition systems (MTS) to model the behaviour of software product lines. Following [5], the vending machine has a European variant serving tea and coffee as well as an american variant serving coffee and cappuccino. The European version accepts euro coins while the US version accepts dollars; in addition, the US version rings a tone when the beverage is served. Its behaviour is modelled by the MTS shown in Figure 2. In an MTS, transitions can be optional (represented by dashed lines), which means that MTS, just as FTS, model a set of different TS. For MTS, these can be obtained by removing certain optional transitions and making the others mandatory. For instance, the European variant can be obtained by removing transitions 1$, cappuccino and ring_a_tone.

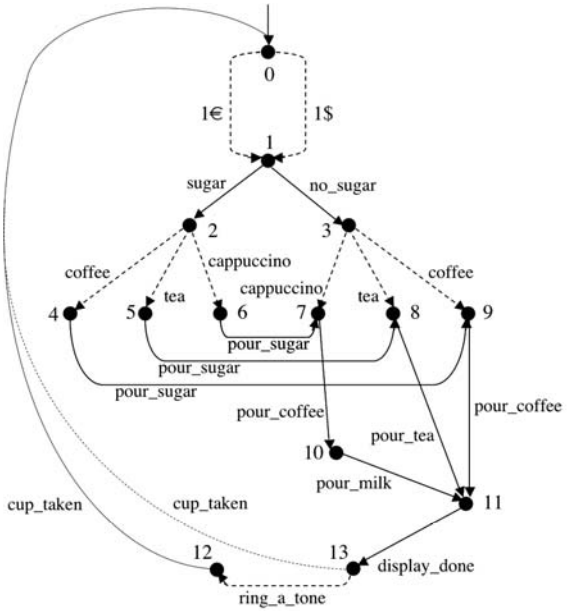


Figure 2: The vending machine MTS, taken from [5].

However, with the MTS given in Figure 2, it is possible to obtain many more variants. For instance, it is possible to obtain a machine which takes euro coins *and* rings a tone when a beverage is served. This would correspond neither to the US nor to the European version; still, it can be easily imagined as a valid system. Moreover, it is also possible to obtain a machine that serves coffee always with sugar, and tea and cappuccino always without; or a machine which would accept dollar and euro coins. These systems are probably not among the systems that the engineer had in mind. The problem that we try to illustrate

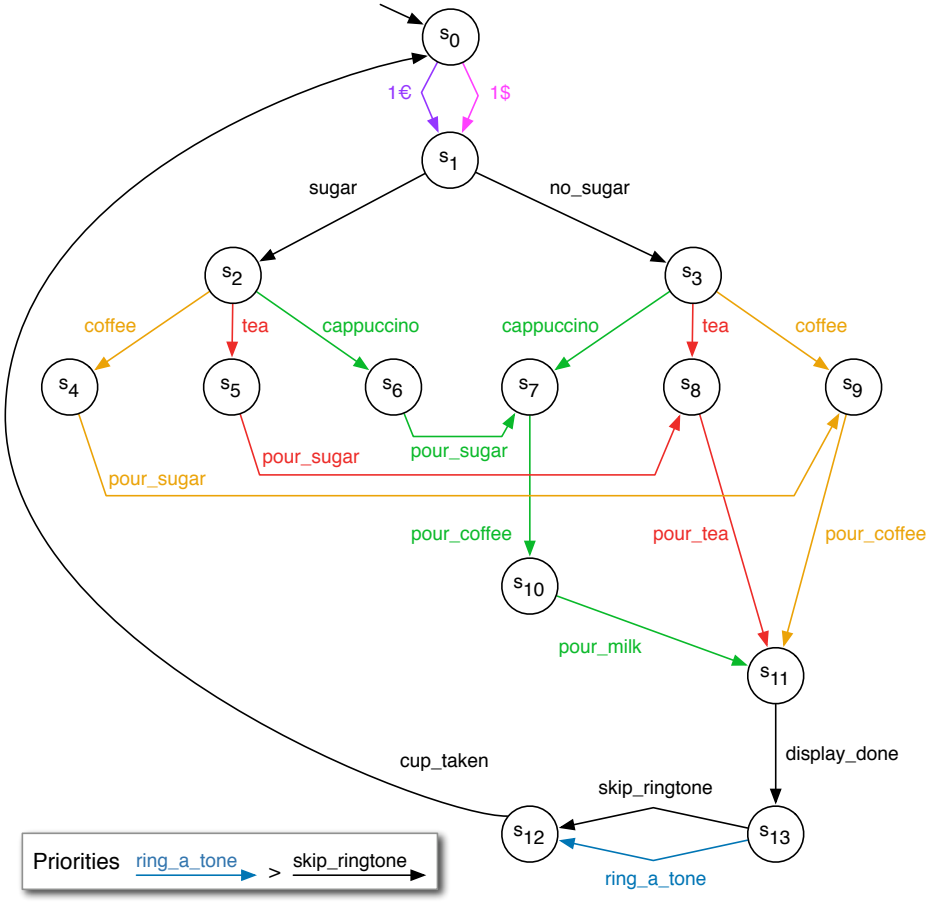


Figure 3: The vending machine modelled as an FTS.

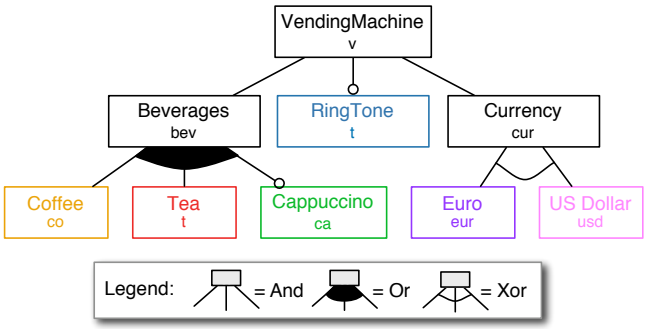


Figure 4: The FD for the vending machine.

here is that in MTS, it is not easily possible to model transitions that belong together so that they always appear together in a system, or not at all (such as the two cappuccino transitions in Figure 2).

The vending machine modelled with an FTS is shown in Figure 3. The FD that comes with the FTS is shown in Figure 4. Valid products of the FD are the US and the European version as described above, but also other variants. In contrast to the MTS of Figure 2, the FTS only models systems that are valid, that is: the pairs of coffee, tea or cappuccino transitions always appear together, because they belong to the same feature. Similarly, the machine will either accept dollars or euros, but not both, since the are modelled as alternatives in the related FD.

4 The wiper system [6]

The car wiper system example was proposed by Gruler *et al.* in [6]. It consists of two subsystems: a sensor unit, able to detect rain, and the wiper itself. Both the sensor and the wiper come in two qualities, high and low. A low quality rain sensor can only distinguish between *rain* and *no rain*, whereas the high quality sensor can also discriminate between *heavy* and *little rain*. Similarly, the high quality wipers can operate at two speeds, whereas the low quality wiper only operates at one speed. In addition, the low quality wipers can be set to wipe permanently. The FD in Figure 5 models this situation.

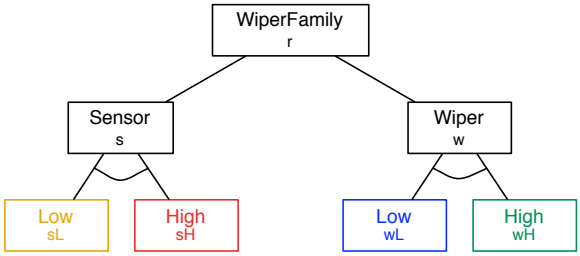


Figure 5: The FD for the original wiper system.

Gruler *et al.* propose PL-CCS [6], a variant of CCS where a new operator $\oplus$ was added to represent alternative choice between two processes. The whole wiper system is modelled with the PL-CCS expression $WipFam$ in Figure 6, i.e. the parallel composition of the sensor and the wiper subsystem. The sensor subsystem is defined as being either the low or the high quality sensor subsystem. The wiper subsystem is defined similarly.

The PL-CCS definition of the two sensor subsystems is given in Figure 7. The low quality sensor will either sense no rain, or it will sense heavy/little rain in which case it sends the message $Rain$. As expected, the high quality sensor behaves differently: in case of heavy rain it sends the message $HvyRain$. An

$$
\begin{aligned}
WipFam &\stackrel{def}{=} Sensor \parallel Wiper \\
Sensor &\stackrel{def}{=} SensL \oplus_1 SensH \\
Wiper &\stackrel{def}{=} WipL \oplus_2 WipH
\end{aligned}
$$

Figure 6: The wiper system in PL-CCS, taken from [6].

Low quality.

$$
\begin{aligned}
SensL &\stackrel{def}{=} non.SensL + ltl.Raining + hvy.Raining + \overline{noRain}.SensL \\
Raining &\stackrel{def}{=} non.SensL + ltl.Raining + hvy.Raining + \overline{rain}.Raining
\end{aligned}
$$

High quality.

$$
\begin{aligned}
SensH &\stackrel{def}{=} non.SensH + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH \\
Medium &\stackrel{def}{=} non.SensH + ltl.Medium + hvy.Heavy + \overline{rain}.Medium \\
Heavy &\stackrel{def}{=} non.SensH + ltl.Medium + hvy.Heavy + \overline{hvyRain}.Heavy
\end{aligned}
$$

Figure 7: The sensor subsystem in PL-CCS, taken from [6].

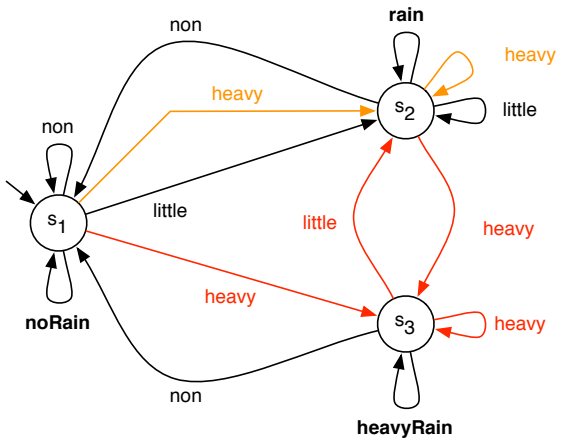


Figure 8: The FTS for the sensor subsystem.

Low quality.

$$
\begin{aligned}
WipL &\stackrel{def}{=} off.WipL + manualOn.Permanent + intvOn.Interval \\
Interval &\stackrel{def}{=} noRain.Interval + intvOff.WipL + intvOn.Interval \\
&\quad + rain.Wiping + hvyRain.Wiping \\
Wiping &\stackrel{def}{=} \overline{slowWip}.Interval + intvOn.Interval \\
Permanent &\stackrel{def}{=} \overline{permWip}.Permanent + off.WipL + intvOn.Interval
\end{aligned}
$$

High quality.

$$
\begin{aligned}
WipH &\stackrel{def}{=} off.WipH + intvOn.AutoIntv \\
AutoIntv &\stackrel{def}{=} noRain.AutoIntv + intvOn.AutoIntv + rain.Slow \\
&\quad + intvOff.WipH + hvyRain.Fast \\
Slow &\stackrel{def}{=} \overline{slowWip}.AutoIntv + intvOn.AutoIntv \\
Fast &\stackrel{def}{=} \overline{fastWip}.AutoIntv + intvOn.AutoIntv
\end{aligned}
$$

Figure 9: The wiper subsystem in PL-CCS, taken from [6].

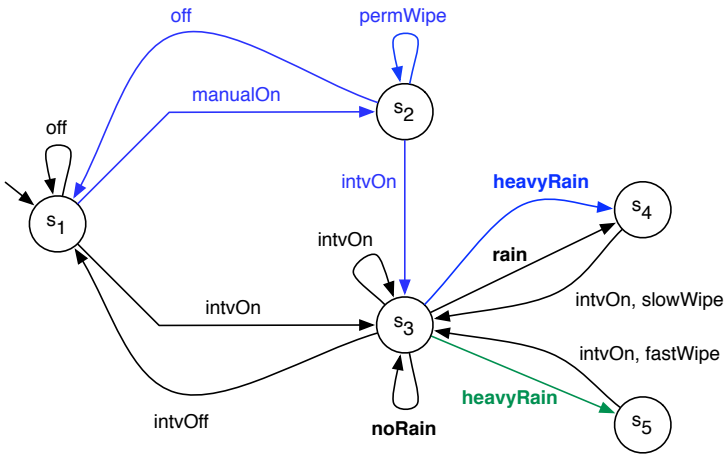


Figure 10: The FTS for the wiper subsystem.

immediate observation is that both subsystems are quite similar and that the sending of the *Rain* message is the same in both cases. Still, the corresponding part has to be duplicated inside both subsystems. An equivalent description in FTS is given in Figure 8. Since the part dealing with the detection of little or no rain is the same for both qualities, the corresponding actions in the FTS are part of the base system instead of being duplicated. Both features visibly differ only in the handling of the heavy rain condition. Note that in Figure 8 and subsequent figures, labels in bold font denote transitions which are synchronised in a parallel composition.

As of the two wiper subsystems, their PL-CCS definition is given in Figure 9. Both subsystems have an interval switch, which will switch interval wiping on. During interval wiping, both subsystems wipe if the sensor subsystem reports rain; the high quality subsystem will wipe faster in case of heavy rain. In addition, the low quality wiper can be set to permanent wiping, which ignores the rain sensor. Here again, both subsystems are almost identical (except for the permanent wiping function), the sole difference being that the high quality variant reacts differently to a *HvyRain* message. As a consequence, the definitions for both subsystems are almost duplicates. This duplication is not needed in FTS. Consider the FTS representation of the wiper subsystem shown in Figure 10. It clearly shows that (except for the permanent wiping function) both versions only differ in their handling of HeavyRain.

We conclude this example with an extension that is not part of the original paper [6]. Consider the case in which the permanent wiping feature can also be supported by high quality wipers (in fact, there is no reason why it should not). That is, permanent wiping will become an individual feature which is optional. A revised FD that accommodates the new feature is presented in Figure 11. To make the corresponding change in the behavioural model, in FTS it is sufficient to relabel the transitions pertaining to the wiping feature, as shown in Figure 12. In PL-CCS, on the other hand, one will have to duplicate the definition of the permanent wiping mode in both subsystem definitions.

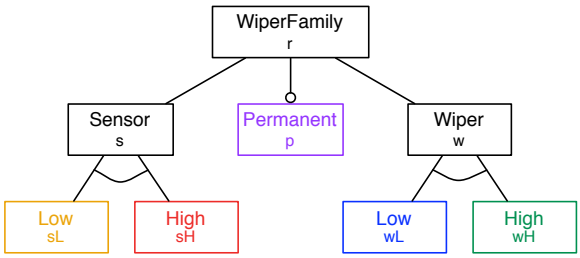


Figure 11: A FD for the wiper system in which permanent wiping is explicitly represented as a feature.

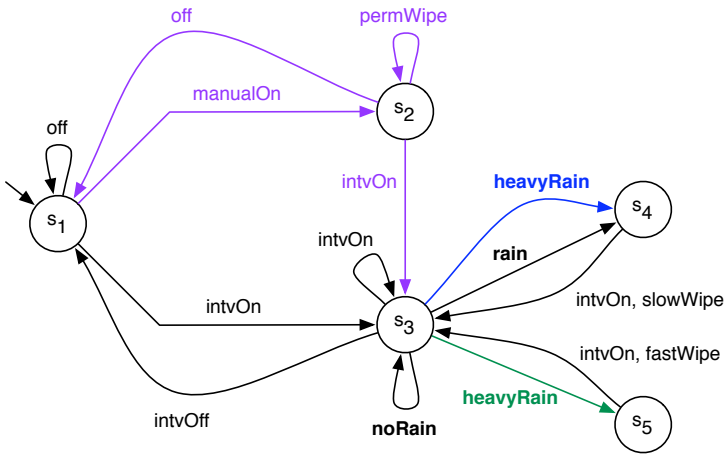


Figure 12: The modified FTS for the wiper system with permanent wiping as a separate feature.

5 The mine pump system [7]

The purpose of the mine pump system [7] is to keep a mine shaft clear of water while avoiding the danger of a methane related explosion. It consists of a water pump, a sensor measuring the water level and a sensor measuring the abundance of methane in the mine. The system is supposed to activate the pump once the water level reaches a preset threshold, but only if the methane is below a critical limit.

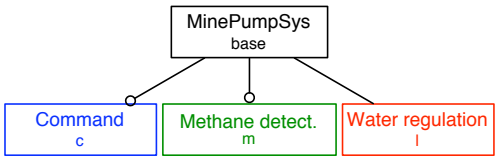


Figure 13: An initial FD for the mine pump controller.

The system consists of three high-level features, shown in Figure 13: (i) a command interface c, which can be used to switch the water regulation function on or off; (ii) a methane alarm interface m, which can receive alarm messages from the methane sensor in case of critical methane, and (iii) the water regulation subsystem l. The system is modelled by the FTS in Figure 14. It maintains a variable representing the system state, and its reactions to events such as a methane alarm or high water depend on this state. In order to keep the description intuitive, the system state is modelled in a separate FTS, shown in Figure 15. Basically, the main FTS only describes the *actions on the system*

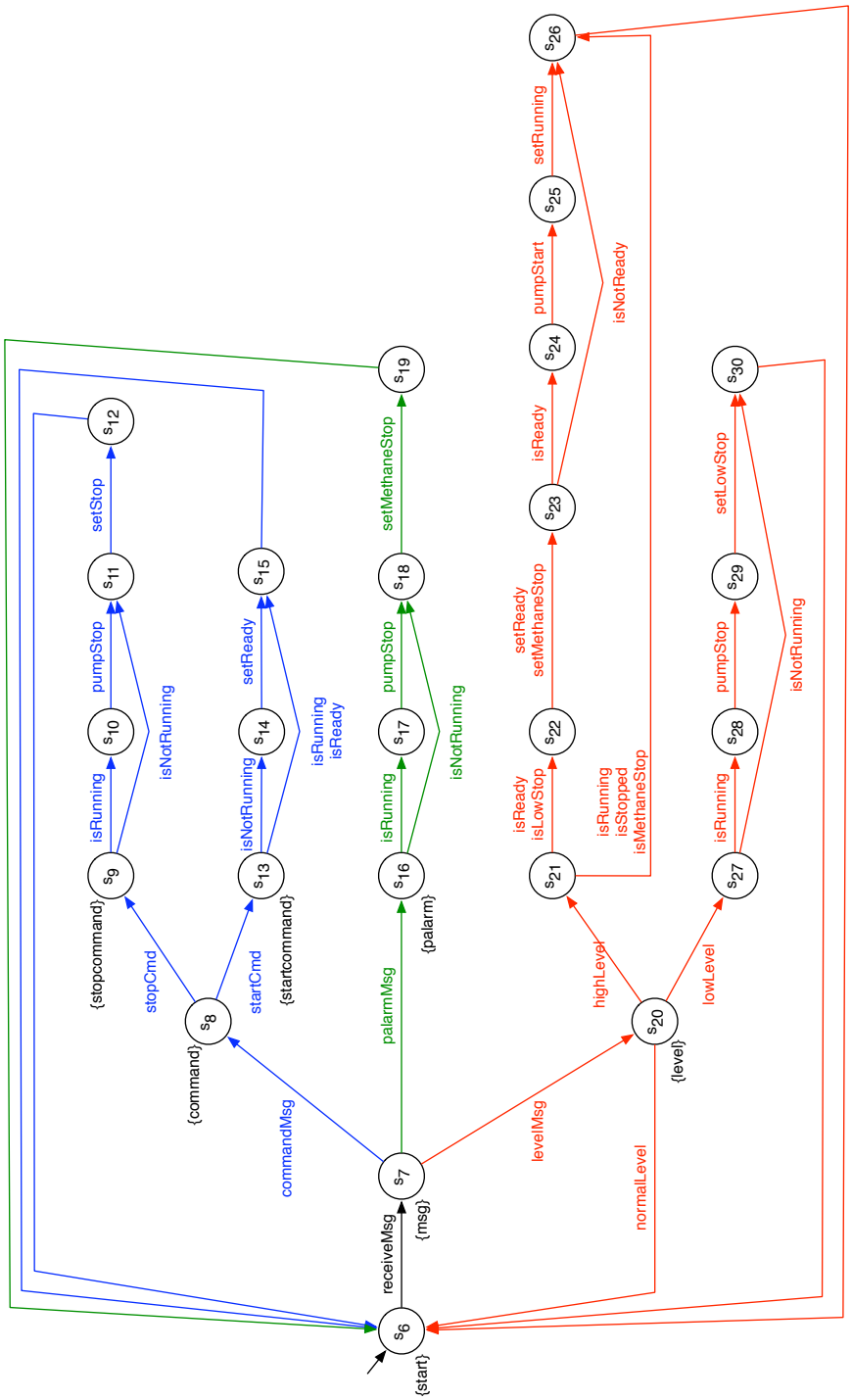


Figure 14: The FTS of the mine pump controller.

state, but does not record it explicitly. The actual FTS of the controller is the parallel composition of these two FTS. Note that in Figure 15 and subsequent figures, transitions represented by dashed lines are transitions that are not labelled by a feature. These transitions will be synchronised during parallel composition and take the feature of the transition with which they are synchronised.

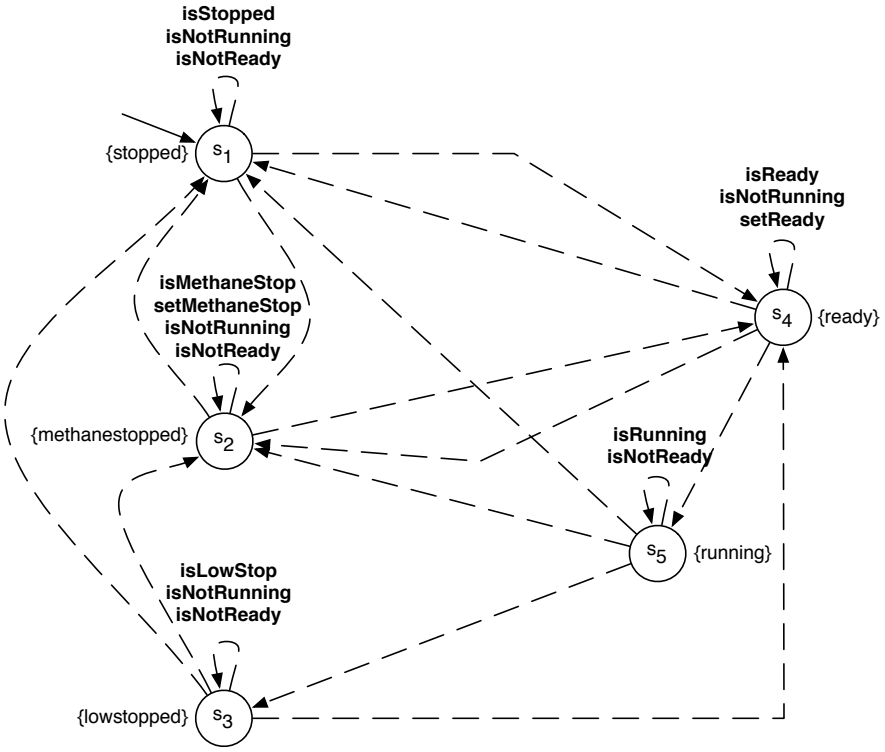


Figure 15: The FTS modelling the system state. The transitions between states are implicitly named setNewState where NewState is the name of the state in which the transition arrives, e.g. setReady.

Intuitively, each time the main FTS executes a set* action, e.g. setReady, it will be synchronised with the corresponding transition in the state FTS. The result is that the state in which the transition arrives is labelled with the new state, e.g. *ready*. The state FTS will thus add an atomic proposition with the system state to each state of the main FTS. This causes a small blowup; the resulting FTS will have hundreds of states.

There are five system states:

- *stopped* means that the water regulation function is off (controlled via the command interface). The system will in no case switch the pump on.

- *ready* means that the water regulation function is on (controlled via the command interface). The system will switch the pump on if there is no methane and the water level is high.
- *running* means that the pump is currently running.
- *lowstopped* means that the pump was stopped because the water level was low. The pump will resume in case the water rises again.
- *methanestopped* means that the pump was stopped because of a critical methane level. The pump will not resume until explicitly told so via the command interface.

The system operates as follows. It will observe three types of events: commands, methane alarm messages and water readings. There are two types of command: stop and start. In case of a start command, the system state is changed and set to *ready*. In case of a stop command, the pump is stopped, and the system state set to *stopped*. In case of a methane alarm, the system stops the pump and sets the system state to *methanestopped*. The system can distinguish between three different water levels: in case of normal water, the system does nothing; if the water is high and the pump not yet running, the system will first check whether it is *ready* or whether it just stopped because of low water (*lowstopped*), if yes, it will check the methane level, and if there is no methane (that is, if after the check it is still *ready*), it will start the pump and set the state to *running*, otherwise it will do nothing. Once the water is low, the system switches off the pump and sets the system state to *lowstopped*.

The system interacts with its environment, which is modelled with three other FTS that are put in parallel with the FTS of the system. The methane level is modelled with the FTS in Figure 16. Methane can rise and lower at will, represented by the methaneRise and methaneLower transitions. The pAlarmMsg and setMethaneStop transitions will synchronise with the system FTS, meaning that the system FTS will receive the alarm message only in case of high methane.

The water pump is represented by the FTS in Figure 17. The pump can be in two states, running or stopped, and the actions pumpStart and pumpStop, synchronised with the system FTS, will cause this state to change. The action pumpRunning is used to model the interaction between the pump and the water. It is synchronised with the water FTS shown in Figure 18: a running pump will

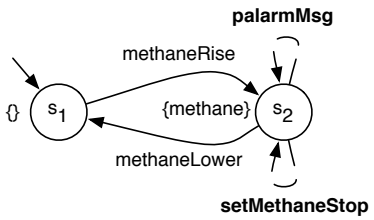


Figure 16: An FTS modelling the environment: the methane level.

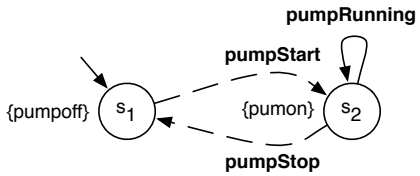


Figure 17: An FTS modelling the environment: the pump.

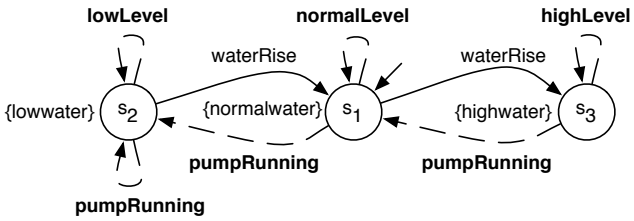


Figure 18: An FTS modelling the environment: the water level.

cause the water level to decrease. The level can rise at will. The low, high and normalLevel actions are synchronised with the main FTS, meaning the system will only observe low, high or normal water if this is indeed the case.

When the command interface and the methane alarm interface are considered optional, as in the first FD in Figure 13, there are four different products. We can add further variability by considering the start and stop message types as well as the three water level readings as individual features. A revised FD is shown in Figure 19. The product line now has 64 products. The revised system FTS is given in Figure 20. The other FTS do not change. Please refer to [3] for a list of LTL properties that were checked for these two FTS as a benchmark of the performance of our model checking procedure.

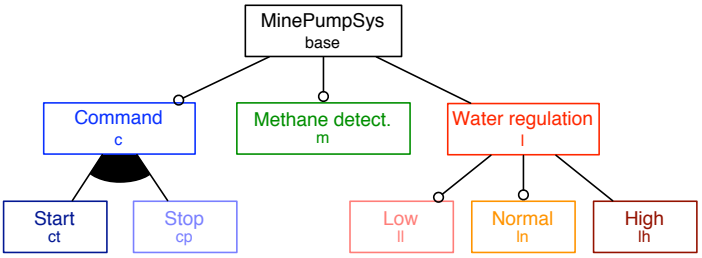


Figure 19: The refined FD for the mine pump system.

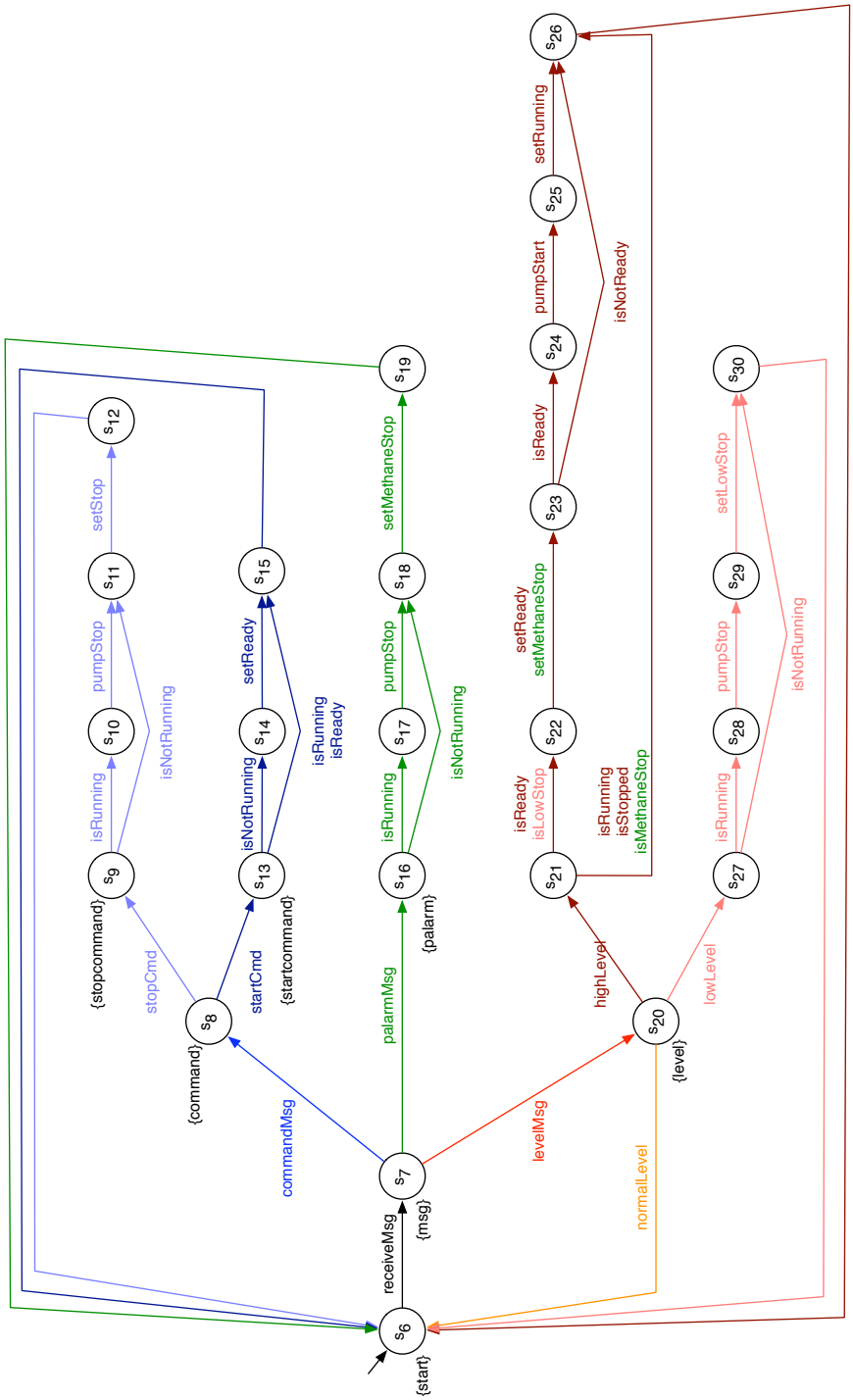


Figure 20: The refined FTS for the mine pump system.

References

[1] C. Baier and J.-P. Katoen. *Principles of Model Checking.* MIT Press, 2007.

[2] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[3] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. Model checking lots of systems: Efficient verification of temporal properties in software product lines. In *32nd International Conference on Software Engineering, ICSE 2010, May 2-8, 2010, Cape Town, South Africa, Proceedings.* IEEE, 2010. To appear.

[4] P. C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns.* SEI Series in Software Engineering. Addison-Wesley, August 2001.

[5] A. Fantechi and S. Gnesi. Formal modeling for product families engineering. In *SPLC 2008*, pages 193–202. IEEE CS, 2008.

[6] A. Gruler, M. Leucker, and K. Scheidemann. Modeling and model checking software product lines. In *IFIP WG 6.1 FMOODS '08*, pages 113–131. Springer, 2008.

[7] J. Kramer, J. Magee, M. Sloman, and A. Lister. Conic: an integrated approach to distributed computer control systems. *Computers and Digital Techniques, IEE Proceedings E*, 130(1):1–10, 1983.

[8] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *RE'06*, pages 139–148, 2006.